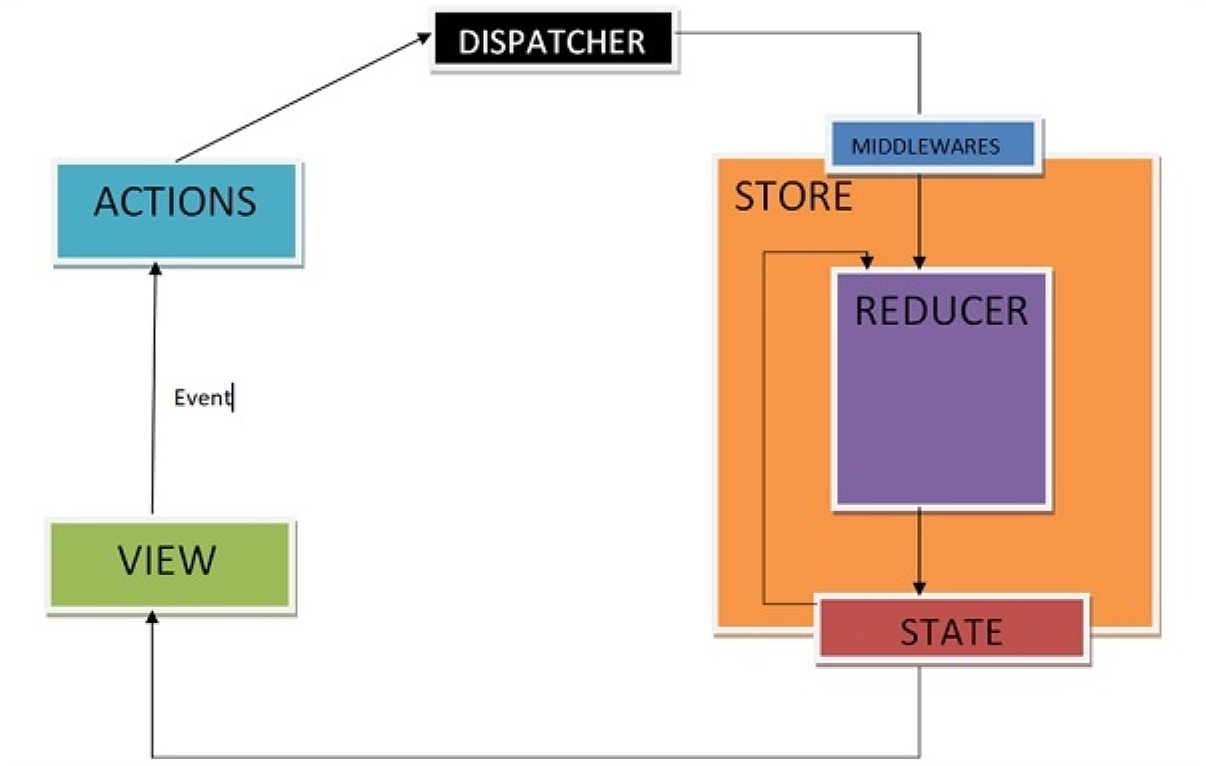
Date=22/12/2020
Lecture By= Akash Handa
Subject ⇒ Redux

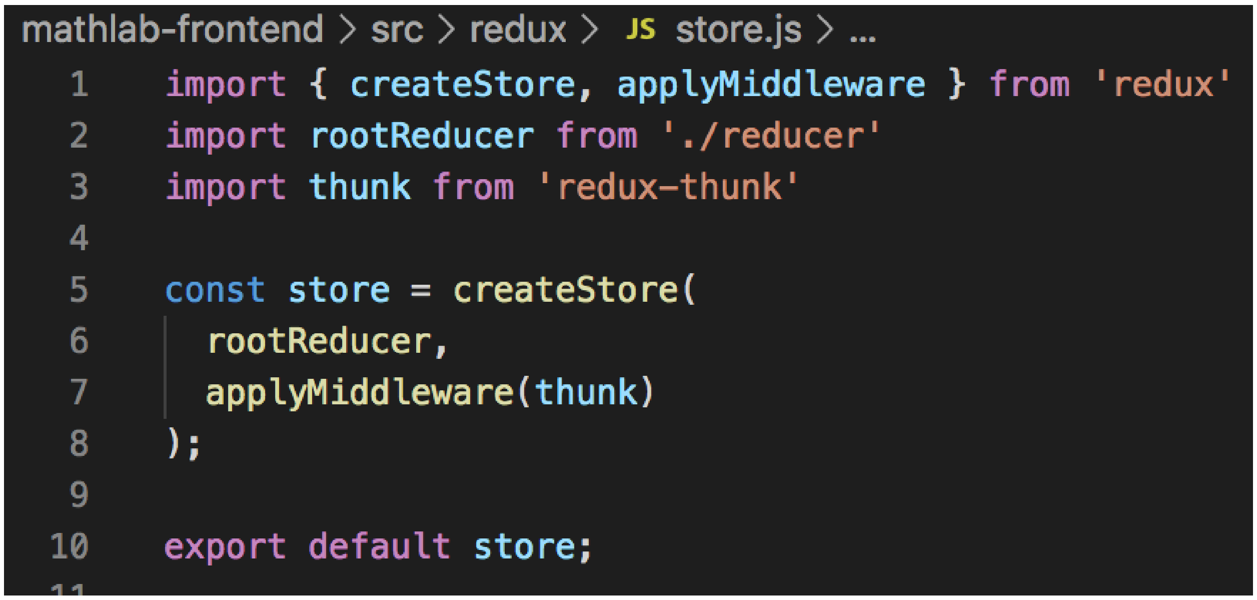
IN PREVIOUS LECTURE (QUICK RECAP) Date-21/12/2020	**In Today's Lecture (Overview)**
What Is Redux?? And Why?? Actions & Action Creators Reducer Selector	applyMiddleware(...middleware) Arguments Tips# Redux Promise

applyMiddleware(...middleware)

Middleware is the suggested way to extend Redux with custom functionality. Middleware lets you wrap the store's dispatch method for fun and profit. The key feature of middleware is that it is composable. Multiple middleware can be combined together, where each middleware requires no knowledge of what comes before or after it in the chain.
The most common use case for middleware is to support asynchronous actions without much boilerplate code or a dependency on a library like Rx. It does so by letting you dispatch async actions in addition to normal actions.
For example, redux-thunk lets the action creators invert control by dispatching functions. They would receive dispatch as an argument and may call it asynchronously. Such functions are called *thunks*. Another example of middleware is redux-promise. It lets you dispatch a Promise async action, and dispatches a normal action when the Promise resolves.
Middleware is not baked into createStore and is not a fundamental part of the Redux architecture, but we consider it useful enough to be supported right in the

core. This way, there is a single standard way to extend dispatch in the ecosystem, and different middleware may compete in expressiveness and utility




mathlab-frontend > src > redux > JS store.js > ...

```
import { createStore, applyMiddleware } from 'redux'
import rootReducer from './reducer'
import thunk from 'redux-thunk'

const store = createStore(
  rootReducer,
  applyMiddleware(thunk)
);

export default store;
```

Arguments

- ...middleware (*arguments*): Functions that conform to the Redux *middleware API*. Each middleware receives Store's dispatch and getState functions as named arguments, and returns a function. That function will be given the next middleware's dispatch method, and is expected to return a function of action calling next(action) with a potentially different argument, or at a different time, or maybe not calling it at all. The last middleware in the chain will receive the real store's [dispatch](#) method as the next parameter, thus ending the chain. So, the middleware signature is ({ getState, dispatch }) => next => action.

Returns

(*Function*) A store enhancer that applies the given middleware. The store enhancer signature is createStore => createStore but the easiest way to apply it is to pass it to createStore() as the last enhancer argument.

Example: Custom Logger Middleware

```
import { createStore, applyMiddleware } from 'redux'
import todos from './reducers'

function logger({ getState }) {
  return next => action => {
    console.log('will dispatch', action)

    // Call the next dispatch method in the middleware chain.
    const returnValue = next(action)

    console.log('state after dispatch', getState())

    // This will likely be the action itself, unless
    // a middleware further in chain changed it.
    return returnValue
  }
}

const store = createStore(todos, ['Use Redux'], applyMiddleware(logger))

store.dispatch({
  type: 'ADD_TODO',
  text: 'Understand the middleware'
})
// (These lines will be logged by the middleware:)
// will dispatch: { type: 'ADD_TODO', text: 'Understand the middleware' }
// state after dispatch: [ 'Use Redux', 'Understand the middleware' ]
```

Tips#

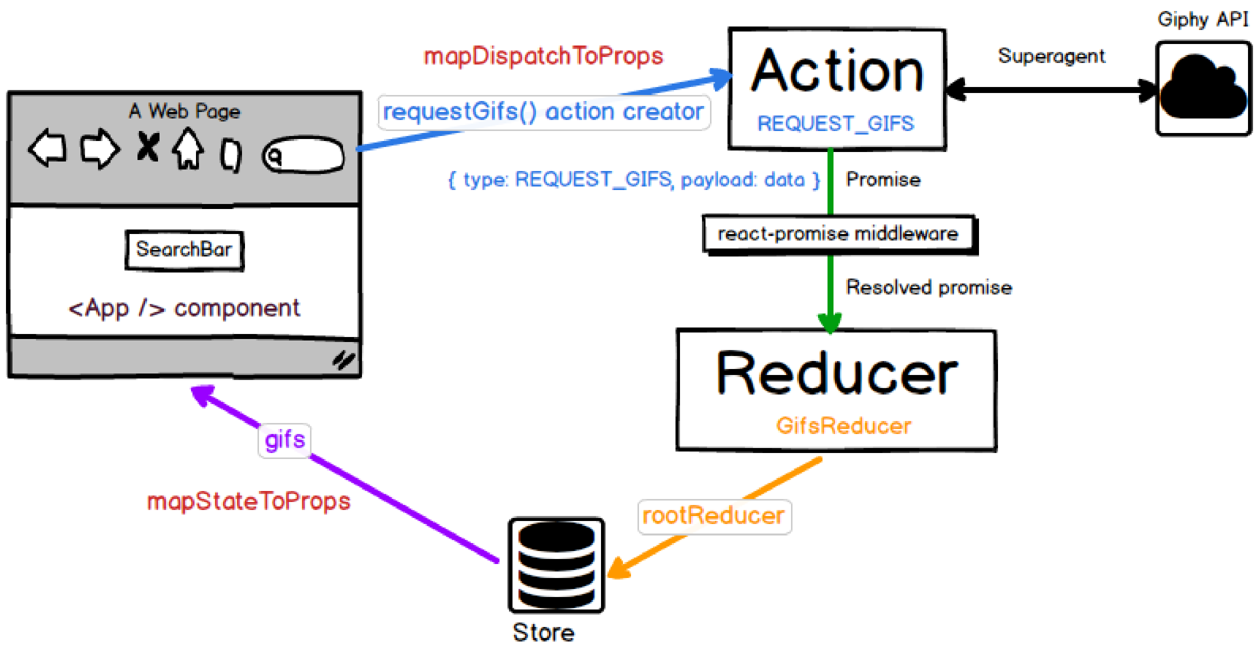
- Middleware only wraps the store's dispatch function. Technically, anything a middleware can do, you can do manually by wrapping every dispatch call, but it's easier to manage this in a single place and define action transformations on the scale of the whole project.
- If you use other store enhancers in addition to applyMiddleware, make sure to put applyMiddleware before them in the composition chain because the middleware is potentially asynchronous. For example, it should go before redux-devtools because otherwise the DevTools won't see the raw actions emitted by the Promise middleware and such.
- If you want to conditionally apply a middleware, make sure to only import it when it's needed:

```
let middleware = [a, b]
if (process.env.NODE_ENV !== 'production') {
  const c = require('some-debug-middleware')
  const d = require('another-debug-middleware')
  middleware = [...middleware, c, d]
}

const store = createStore(
  reducer,
  preloadedState,
  applyMiddleware(...middleware)
)
```

Redux Promise

redux-promise lets you pass **promises** directly to dispatch() , or put **promises** inside of an action object. **redux**-thunk lets you pass functions directly to dispatch() , and makes dispatch() return whatever the thunk function returns (which could be a value, could be a **promise**, or something else)


A Web Page
SearchBar
<App /> component
mapDispatchToProps
requestGifs() action creator
Action
REQUEST_GIFS
Superagent
Giphy API
{ type: REQUEST_GIFS, payload: data }
Promise
react-promise middleware
Resolved promise
Reducer
GifsReducer
rootReducer
Store
gifs
mapStateToProps